Databases

Constraints & Chase

Pierre Senellart

4 March 2026

Outline

FD and MVD

Chase of FD and MVD

Generalization

General chase

References

Setting

- Relational model
- Set semantics
- Named perspective (no order between attributes)
- Untyped attributes
- A relation schema can therefore be seen as a set of attribute names
- All this easily generalizes to other contexts (except bag semantics, which is more complex)

Functional dependency (FD)

- Relation between attributes of the same table R
- Forbids the presence of certain tuples in the database
- Generalizes the notion of a key
- Written $X \to Y$ where X and Y are two sets of attribute names of the same table
- Definition: An instance r satisfies $X \to Y$, written $r \models X \to Y$ if $\forall t, t' \in r$, if $t[X] = t'[X]$, then $t[Y] = t'[Y]$
- Cannot be specified as a constraint in SQL unless they are keys (we will see this is expected: if a relation is in BCNF, no nontrivial FD)

Multivalued dependency (MVD)

- Relation between attributes of the same table R
- Enforces the presence of certain tuples in the table
- Written $X \twoheadrightarrow Y$ where X and Y are two sets of attribute names of the same table
- Definition: An instance r satisfies $X \twoheadrightarrow Y$, written $r \models X \twoheadrightarrow Y$ if $\forall t, t' \in r$, if $t[X] = t'[X]$, and Z is the set of attributes of R outside $X \cup Y$, then there exists a tuple t'' such that $t''[X] = t[X]$, $t''[Y] = t[Y]$ and $t''[Z] = t'[Z]$
- Cannot be specified as a constraint in SQL (we will see this is expected: if a relation is in 4NF, no nontrivial MVD)

What are FD and MVD for?

- Like all constraints: they impose logical constraints at the schema level on the database, ensuring data integrity
- They allow defining a theory of normalization which connects to practical database design concerns (cf. next lecture)
- Important special cases of a family of dependencies (EGD+TGD) that appear in many contexts (see later)

Outline

FD and MVD

Chase of FD and MVD
- Three examples
- Algorithm
- Application

Generalization

General chase

References

The chase

Let R be a relation schema, and let Σ be a set of functional and multivalued dependencies on R.

The chase is an algorithm that decides whether a functional or multivalued dependency is implied by Σ in R:

$$\Sigma \overset{?}{\models} \sigma$$

In other words:

$$\forall r \text{ instance of } R \quad r \models \Sigma \quad \overset{?}{\Rightarrow} \quad r \models \sigma$$

Outline

FD and MVD

Chase of FD and MVD

- Three examples
- Algorithm
- Application

Generalization

General chase

References

Examples

We consider the schema $\{A, B, C, D\}$.

1. $\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \to \{C\}\} \stackrel{?}{\models} \{A\} \to \{C\}$

Examples

We consider the schema $\{A, B, C, D\}$.

1. $\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \to \{C\}\} \stackrel{?}{\models} \{A\} \to \{C\}$
2. $\{\{A\} \twoheadrightarrow \{B\}, \{B\} \twoheadrightarrow \{C\}\} \stackrel{?}{\models} \{A\} \twoheadrightarrow \{C\}$

Examples

We consider the schema $\{A, B, C, D\}$.

1. $\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \rightarrow \{C\}\} \stackrel{?}{\models} \{A\} \rightarrow \{C\}$
2. $\{\{A\} \twoheadrightarrow \{B\}, \{B\} \twoheadrightarrow \{C\}\} \stackrel{?}{\models} \{A\} \twoheadrightarrow \{C\}$
3. $\{\{A\} \twoheadrightarrow \{B, C\}, \{C, D\} \rightarrow \{B\}\} \stackrel{?}{\models} \{A\} \rightarrow \{B\}$

Example 1 (1/6)

$$\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \rightarrow \{C\}\} \stackrel{?}{\models} \{A\} \rightarrow \{C\}$$

Create an instance r on the schema $\{A, B, C, D\}$ with two tuples and distinct values for all attributes.

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2

Example 1 (2/6)

$$\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \to \{C\}\} \stackrel{?}{\models} \{A\} \to \{C\}$$

We want to determine whether $\{A\} \to \{C\}$.
Make all values of A identical.

$$a_1 = a_2$$

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2

Example 1 (3/6)

$$\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \to \{C\}\} \stackrel{?}{\models} \{A\} \to \{C\}$$

Use $\{A\} \twoheadrightarrow \{B, C\}$. Create two new tuples by copying the two tuples with the same value of A but swapping their values for $\{B, C\}$. The multivalued dependency generates tuples. This is a tuple-generating dependency (TGD).

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_2	c_2	d_1
a_1	b_1	c_1	d_2

Example 1 (4/6)

$$\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \to \{C\}\} \stackrel{?}{\models} \{A\} \to \{C\}$$

Use $\{D\} \to \{C\}$. For every two tuples with the same value of D, make their value of C identical: $c_1 = c_2$. The functional dependency generates equalities. This is an equality-generating dependency (EGD).

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_1	b_2	c_1	d_1
a_1	b_1	c_1	d_2

Example 1 (5/6)

$$\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \to \{C\}\} \stackrel{?}{\models} \{A\} \to \{C\}$$

No rule applies: the relation already satisfies the two dependencies of Σ. We observe that r satisfies $\{A\} \to \{C\}$.

$$r \models \{A\} \to \{C\}$$

The answer is therefore yes.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_1	b_2	c_1	d_1
a_1	b_1	c_1	d_2

Example 1 (6/6)

r also satisfies $\{D\} \to \{A\}$ but this is a coincidence. We can only answer the question about $\{A\} \to \{C\}$.
Another chase would be necessary for $\{D\} \to \{A\}$.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_1	b_2	c_1	d_1
a_1	b_1	c_1	d_2

Example 2 (1/5)

$$R = \{A, B, C, D\}$$

$$\{\{A\} \twoheadrightarrow \{B\}, \{B\} \twoheadrightarrow \{C\}\} \stackrel{?}{\models} \{A\} \twoheadrightarrow \{C\}$$

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2

Example 2 (2/5)

$$R = \{A, B, C, D\}$$

$$\{\{A\} \twoheadrightarrow \{B\}, \{B\} \twoheadrightarrow \{C\}\} \stackrel{?}{\models} \{A\} \twoheadrightarrow \{C\}$$

We want to determine whether $\{A\} \twoheadrightarrow \{C\}$.
We make all values of A identical.

$$a_1 = a_2$$

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2

Example 2 (3/5)

$$R = \{A, B, C, D\}$$

$$\{\{A\} \twoheadrightarrow \{B\}, \{B\} \twoheadrightarrow \{C\}\} \stackrel{?}{\models} \{A\} \twoheadrightarrow \{C\}$$

Use $\{A\} \twoheadrightarrow \{B\}$.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_2	c_1	d_1
a_1	b_1	c_2	d_2

Example 2 (4/5)

$$R = \{A, B, C, D\}$$

$$\{\{A\} \twoheadrightarrow \{B\}, \{B\} \twoheadrightarrow \{C\}\} \overset{?}{\models} \{A\} \twoheadrightarrow \{C\}$$

Use $\{B\} \twoheadrightarrow \{C\}$ (twice).

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_2	c_1	d_1
a_1	b_1	c_2	d_2
a_1	b_1	c_2	d_1
a_1	b_1	c_1	d_2
a_1	b_2	c_1	d_2
a_1	b_2	c_2	d_1

Example 2 (5/5)

No rule applies.

$$r \not\models \{A\} \twoheadrightarrow \{C\}$$

The answer is therefore yes.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_2	c_1	d_1
a_1	b_1	c_2	d_2
a_1	b_1	c_2	d_1
a_1	b_1	c_1	d_2
a_1	b_2	c_1	d_2
a_1	b_2	c_2	d_1

Example 3 (1/3)

$$\{\{A\} \twoheadrightarrow \{B, C\}, \{C, D\} \rightarrow \{B\}\} \stackrel{?}{\models} \{A\} \rightarrow \{B\}$$

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2

Example 3 (2/3)

$$\{\{A\} \twoheadrightarrow \{B, C\}, \{C, D\} \rightarrow \{B\}\} \stackrel{?}{\models} \{A\} \rightarrow \{B\}$$

Use $\{A\} \twoheadrightarrow \{B, C\}$.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_2	c_2	d_1
a_1	b_1	c_1	d_2

Example 3 (3/3)

No rule applies.

$$r \not\models \{A\} \to \{B\}$$

The answer is therefore no.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_2	c_2	d_1
a_1	b_1	c_1	d_2

We have constructed a counterexample.

Outline

FD and MVD

Chase of FD and MVD

- Three examples
- Algorithm
- Application

Generalization

General chase

References

The power of the chase

The chase is a very powerful tool: it can be used to prove that a functional or multivalued dependency is satisfied!

The power of the chase

The chase is a very powerful tool: it can be used to prove that a functional or multivalued dependency is satisfied!

Theorem ([Maier et al., 1979])

The chase with FDs and MVDs always terminates after a finite number of operations, and constructs a counterexample if and only if one exists.

The power of the chase

The chase is a very powerful tool: it can be used to prove that a functional or multivalued dependency is satisfied!

Theorem ([Maier et al., 1979])

The chase with FDs and MVDs always terminates after a finite number of operations, and constructs a counterexample if and only if one exists.

Why? We will see that this is a special case of a much more general result.

Starting the chase

Let Σ be a set of functional and multivalued dependencies on a relation schema R. Let σ be a functional or multivalued dependency.

$$\sigma = X \to Y \text{ or } \sigma = X \twoheadrightarrow Y$$

Starting the chase

Let Σ be a set of functional and multivalued dependencies on a relation schema R. Let σ be a functional or multivalued dependency.

$$\sigma = X \to Y \text{ or } \sigma = X \twoheadrightarrow Y$$

1. Create a table r of schema R with two tuples with all values different.

Starting the chase

Let Σ be a set of functional and multivalued dependencies on a relation schema R. Let σ be a functional or multivalued dependency.

$$\sigma = X \to Y \text{ or } \sigma = X \twoheadrightarrow Y$$

1. Create a table r of schema R with two tuples with all values different.
2. For each $A \in X$, make the values of A identical (by choosing different values for each A).

Chase

Repeat until reaching a fixed point:

1. For each functional dependency $Z \to V \in \Sigma$:
   - If two tuples have the same value of Z, make their value of V identical.
2. For each multivalued dependency $Z \twoheadrightarrow V \in \Sigma$:
   - If two tuples have the same value of Z, add two new tuples with the same values, except that the values of V are swapped.

At the fixed point (which we always reach):

$$r \models \sigma \quad \text{is equivalent to} \quad \Sigma \models \sigma$$

It suffices to check whether r satisfies σ.

Outline

FD and MVD

Chase of FD and MVD

- Three examples
- Algorithm
- Application

Generalization

General chase

References

Testing whether a decomposition is lossless

If $R = X \cup Y \cup Z$ with X, Y, Z disjoint, an MVD $X \twoheadrightarrow Y$ is satisfied by r iff:

$$\pi_{X \cup Y}(r) \bowtie \pi_{X \cup Z}(r) = r$$

We can therefore use MVDs to test whether a relation schema $R = X \cup Y \cup Z$ (satisfying dependencies Σ) can be decomposed into two relations $R_1 = X \cup Y$ and $R_2 = X \cup Z$ losslessly:

Testing whether a decomposition is lossless

If $R = X \cup Y \cup Z$ with X, Y, Z disjoint, an MVD $X \twoheadrightarrow Y$ is satisfied by r iff:

$$\pi_{X \cup Y}(r) \bowtie \pi_{X \cup Z}(r) = r$$

We can therefore use MVDs to test whether a relation schema $R = X \cup Y \cup Z$ (satisfying dependencies Σ) can be decomposed into two relations $R_1 = X \cup Y$ and $R_2 = X \cup Z$ losslessly:

Use the chase to decide whether $\Sigma \models^{?} X \twoheadrightarrow Y$

(or, equivalently, $\Sigma \models^{?} X \twoheadrightarrow Z$)

Outline

FD and MVD

Chase of FD and MVD

Generalization
- EGD and TGD
- Inclusion dependencies (ID)

General chase

References

Outline

FD and MVD

Chase of FD and MVD

Generalization

- EGD and TGD
- Inclusion dependencies (ID)

General chase

References

FD as logical dependencies

$R = (A_1, \ldots A_n)$; $\{A_1\} \rightarrow \{A_2 A_3\}$ can be expressed as:

$$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n \forall a'_2 \forall a'_3 \ldots \forall a'_n$$

$$R(a_1, a_2, a_3, a_4, \ldots, a_n) \wedge R(a_1, a'_2, a'_3, a'_4, \ldots, a'_n) \Rightarrow a_2 = a'_2 \wedge a_3 = a'_3$$

FD as logical dependencies

$R = (A_1, \ldots A_n)$; $\{A_1\} \rightarrow \{A_2 A_3\}$ can be expressed as:

$$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n \forall a'_2 \forall a'_3 \ldots \forall a'_n$$
$$R(a_1, a_2, a_3, a_4, \ldots, a_n) \wedge R(a_1, a'_2, a'_3, a'_4, \ldots, a'_n) \Rightarrow a_2 = a'_2 \wedge a_3 = a'_3$$

Such a formula is an instance of a more general formula:

$$\forall x_1 \ldots \forall x_m \quad \varphi(x_1 \ldots x_m) \Rightarrow \psi(x_1 \ldots x_m)$$

where $\varphi(x_1 \ldots x_m)$ is a CQ with variables $x_1 \ldots x_m$ and $\psi(x_1 \ldots x_m)$ is a conjunction of equalities with variables $x_1 \ldots x_m$.

FD as logical dependencies

$R = (A_1, \ldots A_n)$; $\{A_1\} \rightarrow \{A_2 A_3\}$ can be expressed as:

$$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n \forall a'_2 \forall a'_3 \ldots \forall a'_n$$
$$R(a_1, a_2, a_3, a_4, \ldots, a_n) \wedge R(a_1, a'_2, a'_3, a'_4, \ldots, a'_n) \Rightarrow a_2 = a'_2 \wedge a_3 = a'_3$$

Such a formula is an instance of a more general formula:

$$\forall x_1 \ldots \forall x_m \quad \varphi(x_1 \ldots x_m) \Rightarrow \psi(x_1 \ldots x_m)$$

where $\varphi(x_1 \ldots x_m)$ is a CQ with variables $x_1 \ldots x_m$ and $\psi(x_1 \ldots x_m)$ is a conjunction of equalities with variables $x_1 \ldots x_m$.

Such formulas are called equality-generating dependencies (EGDs).

FD as logical dependencies

$R = (A_1, \ldots A_n)$; $\{A_1\} \rightarrow \{A_2 A_3\}$ can be expressed as:

$$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n \forall a'_2 \forall a'_3 \ldots \forall a'_n$$
$$R(a_1, a_2, a_3, a_4, \ldots, a_n) \wedge R(a_1, a'_2, a'_3, a'_4, \ldots, a'_n) \Rightarrow a_2 = a'_2 \wedge a_3 = a'_3$$

Such a formula is an instance of a more general formula:

$$\forall x_1 \ldots \forall x_m \quad \varphi(x_1 \ldots x_m) \Rightarrow \psi(x_1 \ldots x_m)$$

where $\varphi(x_1 \ldots x_m)$ is a CQ with variables $x_1 \ldots x_m$ and $\psi(x_1 \ldots x_m)$ is a conjunction of equalities with variables $x_1 \ldots x_m$.

Such formulas are called equality-generating dependencies (EGDs).

FDs have the particularity of having:

- only 2 atoms in the left-hand side
- only 1 relation mentioned

MVD as logical dependencies

$R = (A_1, \ldots A_n)$; $\{A_1\} \twoheadrightarrow \{A_2 A_3\}$ can be expressed as:

$$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n \forall a'_2 \forall a'_3 \ldots \forall a'_n$$

$$R(a_1, a_2, a_3, \ldots, a_n) \wedge R(a_1, a'_2, a'_3, \ldots, a'_n) \Rightarrow R(a_1, a'_2, a'_3, a_4, \ldots, a_n)$$

MVD as logical dependencies

$R = (A_1, \ldots A_n)$; $\{A_1\} \twoheadrightarrow \{A_2 A_3\}$ can be expressed as:

$$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n \forall a'_2 \forall a'_3 \ldots \forall a'_n$$
$$R(a_1, a_2, a_3, \ldots, a_n) \wedge R(a_1, a'_2, a'_3, \ldots, a'_n) \Rightarrow R(a_1, a'_2, a'_3, a_4, \ldots, a_n)$$

Such a formula is an instance of a more general formula:

$$\forall x_1 \ldots \forall x_m \quad \varphi(x_1 \ldots x_m) \Rightarrow \exists y_1 \ldots \exists y_p \, \psi(x_1 \ldots x_m, y_1, \ldots, y_p)$$

where $\varphi(x_1 \ldots x_m)$ is a CQ with variables $x_1 \ldots x_m$ and $\psi(x_1 \ldots x_m, y_1, \ldots, y_p)$ is a CQ with variables $x_1 \ldots x_m, y_1 \ldots y_p$.

MVD as logical dependencies

$R = (A_1, \ldots A_n)$; $\{A_1\} \twoheadrightarrow \{A_2 A_3\}$ can be expressed as:

$$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n \forall a'_2 \forall a'_3 \ldots \forall a'_n$$
$$R(a_1, a_2, a_3, \ldots, a_n) \wedge R(a_1, a'_2, a'_3, \ldots, a'_n) \Rightarrow R(a_1, a'_2, a'_3, a_4, \ldots, a_n)$$

Such a formula is an instance of a more general formula:

$$\forall x_1 \ldots \forall x_m \quad \varphi(x_1 \ldots x_m) \Rightarrow \exists y_1 \ldots \exists y_p \, \psi(x_1 \ldots x_m, y_1, \ldots, y_p)$$

where $\varphi(x_1 \ldots x_m)$ is a CQ with variables $x_1 \ldots x_m$ and $\psi(x_1 \ldots x_m, y_1, \ldots, y_p)$ is a CQ with variables $x_1 \ldots x_m, y_1 \ldots y_p$.
Such formulas are called tuple-generating dependencies (TGDs).

MVD as logical dependencies

$R = (A_1, \ldots A_n)$; $\{A_1\} \twoheadrightarrow \{A_2 A_3\}$ can be expressed as:

$$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n \forall a'_2 \forall a'_3 \ldots \forall a'_n$$
$$R(a_1, a_2, a_3, \ldots, a_n) \wedge R(a_1, a'_2, a'_3, \ldots, a'_n) \Rightarrow R(a_1, a'_2, a'_3, a_4, \ldots, a_n)$$

Such a formula is an instance of a more general formula:

$$\forall x_1 \ldots \forall x_m \quad \varphi(x_1 \ldots x_m) \Rightarrow \exists y_1 \ldots \exists y_p \, \psi(x_1 \ldots x_m, y_1, \ldots, y_p)$$

where $\varphi(x_1 \ldots x_m)$ is a CQ with variables $x_1 \ldots x_m$ and $\psi(x_1 \ldots x_m, y_1, \ldots, y_p)$ is a CQ with variables $x_1 \ldots x_m, y_1 \ldots y_p$.
Such formulas are called tuple-generating dependencies (TGDs).
MVDs have the particularity of having:

- only 2 atoms in the left-hand side (more than 2: JD)
- only 1 atom in the right-hand side, no $\exists$
- only one relation mentioned

Outline

FD and MVD

Chase of FD and MVD

Generalization

- EGD and TGD
- Inclusion dependencies (ID)

General chase

References

Inclusion dependencies

- FDs and MVDs can only express constraints (or dependencies) that are local to a relation

Inclusion dependencies

- FDs and MVDs can only express constraints (or dependencies) that are local to a relation
- In practice, we want to express dependencies between relations, in particular foreign keys

Inclusion dependencies

- FDs and MVDs can only express constraints (or dependencies) that are local to a relation
- In practice, we want to express dependencies between relations, in particular foreign keys
- The tuples of values of some attributes are included in the values of other attributes (for example of another table)

Inclusion dependencies

- FDs and MVDs can only express constraints (or dependencies) that are local to a relation
- In practice, we want to express dependencies between relations, in particular foreign keys
- The tuples of values of some attributes are included in the values of other attributes (for example of another table)
- In SQL:

```
CREATE TABLE R1(
  A INT, B VARCHAR(42), ...,
  FOREIGN KEY (A,B) REFERENCES R2(X,Y) )
```

Inclusion dependencies

- FDs and MVDs can only express constraints (or dependencies) that are local to a relation
- In practice, we want to express dependencies between relations, in particular foreign keys
- The tuples of values of some attributes are included in the values of other attributes (for example of another table)
- In SQL:

```
CREATE TABLE R1(
  A INT, B VARCHAR(42), ...,
  FOREIGN KEY (A,B) REFERENCES R2(X,Y) )
```

- Notation: $R_1[A, B] \subseteq R_2[X, Y]$

ID as logical dependencies

$R = (A_1, \ldots A_n)$, $S = (B_1, \ldots B_m)$; $R[A_1] \subseteq S[B_2]$ can be expressed as:

$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n$

$$R(a_1, a_2, a_3, \ldots, a_n) \Rightarrow \exists b_1 \exists b_3 \ldots \exists b_m \; S(b_1, a_1, b_3, \ldots, b_m)$$

ID as logical dependencies

$R = (A_1, \ldots A_n)$, $S = (B_1, \ldots B_m)$; $R[A_1] \subseteq S[B_2]$ can be expressed as:

$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n$

$$R(a_1, a_2, a_3, \ldots, a_n) \Rightarrow \exists b_1 \exists b_3 \ldots \exists b_m \; S(b_1, a_1, b_3, \ldots, b_m)$$

Such a formula is an instance of a TGD:

$$\forall x_1 \ldots \forall x_m \quad \varphi(x_1 \ldots x_m) \Rightarrow \exists y_1 \ldots \exists y_p \, \psi(x_1 \ldots x_m, y_1, \ldots, y_p)$$

ID as logical dependencies

$R = (A_1, \ldots A_n)$, $S = (B_1, \ldots B_m)$; $R[A_1] \subseteq S[B_2]$ can be expressed as:

$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n$

$$R(a_1, a_2, a_3, \ldots, a_n) \Rightarrow \exists b_1 \exists b_3 \ldots \exists b_m \; S(b_1, a_1, b_3, \ldots, b_m)$$

Such a formula is an instance of a TGD:

$$\forall x_1 \ldots \forall x_m \quad \varphi(x_1 \ldots x_m) \Rightarrow \exists y_1 \ldots \exists y_p \; \psi(x_1 \ldots x_m, y_1, \ldots, y_p)$$

IDs have the particularity of having:

- only 1 atom on the left-hand side
- only 1 atom on the right-hand side

Outline

FD and MVD

Chase of FD and MVD

Generalization

General chase

References

General chase

- The chase is not limited to FDs and MVDs
- In fact, it can be defined for an arbitrary set of TGDs and EGDs, which goes beyond FDs, MVDs, JDs, and even IDs
- TGDs (and EGDs) are a fundamental research object, beyond database constraints. Known in other communities as *existential rules* [Baget et al., 2011], *Datalog*$^{\pm}$ [Calì et al., 2010]
- The chase is a major tool, in practice and in theory, for working with these objects

Starting the chase

Let Σ be a set of TGDs and EGDs over a relational schema $\mathcal{S}$.

Let σ be a TGD or EGD. We want to determine $\Sigma \overset{?}{\models} \sigma$.

Starting the chase

Let Σ be a set of TGDs and EGDs over a relational schema $\mathcal{S}$.

Let σ be a TGD or EGD. We want to determine $\Sigma \overset{?}{\models} \sigma$.

1. Create a database d over schema $\mathcal{S}$ whose tuples are the atoms appearing in the left-hand side of σ (or, more specifically, the canonical database associated to the CQ formed of the left-hand side of σ)

Starting the chase

Let Σ be a set of TGDs and EGDs over a relational schema $\mathcal{S}$.

Let σ be a TGD or EGD. We want to determine $\Sigma \stackrel{?}{\models} \sigma$.

1. Create a database d over schema $\mathcal{S}$ whose tuples are the atoms appearing in the left-hand side of σ (or, more specifically, the canonical database associated to the CQ formed of the left-hand side of σ)
2. Ensure that the variables shared between these atoms of σ have the same value in the initial database (and that non-shared variables have distinct values).

Chase

Repeat until reaching a fixed point:

1. For each EGD:
   - If the CQ on the left-hand side is satisfied in the database, enforce the equalities on the right-hand side on the values in the database.
2. For each TGD:
   - If the CQ on the left-hand side is satisfied in the database, but not the one on the right-hand side, add to the database the tuples implied by the right-hand side. For existentially quantified variables, introduce new values (always fresh, never used; we call them nulls).

At the fixed point (which we may fail to reach):

$$d \models \sigma \quad \text{is equivalent to} \quad \Sigma \models \sigma$$

When do we have termination?

Theorem

If there is no existentially quantified variable in the dependencies, the chase terminates.

When do we have termination?

Theorem

If there is no existentially quantified variable in the dependencies, the chase terminates.

Proof.

Since we cannot introduce new values, the chase can produce only a bounded number of tuples. □

When do we have termination?

Theorem

If there is no existentially quantified variable in the dependencies, the chase terminates.

Proof.

Since we cannot introduce new values, the chase can produce only a bounded number of tuples. □

Can be generalized to other settings: when TGDs do not introduce cycles, or when they introduce only "nice" cycles (e.g., weak acyclicity [Fagin et al., 2005]). But fails if we have arbitrary FDs and IDs.

When do we have termination?

Theorem

If there is no existentially quantified variable in the dependencies, the chase terminates.

Proof.

Since we cannot introduce new values, the chase can produce only a bounded number of tuples. □

Can be generalized to other settings: when TGDs do not introduce cycles, or when they introduce only "nice" cycles (e.g., weak acyclicity [Fagin et al., 2005]). But fails if we have arbitrary FDs and IDs. In fact, implication of a dependency by a collection of FDs and IDs is undecidable. [Chandra et al., 1981]

Why is the chase important?

Theorem ([Fagin et al., 2005])

The result of the chase is a universal model: all databases satisfying the dependencies have a subinstance that is homomorphic to the chase result. In other words, a CQ is true on the chase result if and only if it is true on all databases satisfying the dependencies.

Why is the chase important?

Theorem ([Fagin et al., 2005])

The result of the chase is a universal model: all databases satisfying the dependencies have a subinstance that is homomorphic to the chase result. In other words, a CQ is true on the chase result if and only if it is true on all databases satisfying the dependencies.

- Note that the chase result is not unique (depends on the application order), but works for any result
- When the chase does not terminate, it can still be useful: e.g., we can sometimes restrict ourselves to a bounded portion of the chase

Proof idea: maintain a homomorphism along the chase

Consider a chase sequence (finite or infinite):

$$I_0 \to I_1 \to I_2 \to \cdots \qquad \text{and} \qquad I_\infty = \bigcup_{k \geq 0} I_k.$$

Proof idea: maintain a homomorphism along the chase

Consider a chase sequence (finite or infinite):

$$I_0 \to I_1 \to I_2 \to \cdots \qquad \text{and} \qquad I_\infty = \bigcup_{k \geq 0} I_k.$$

Fix any instance J with $J \models \Sigma$ and a homomorphism $h_0 : I_0 \to J$.

Proof idea: maintain a homomorphism along the chase

Consider a chase sequence (finite or infinite):

$$I_0 \to I_1 \to I_2 \to \cdots \qquad \text{and} \qquad I_\infty = \bigcup_{k \geq 0} I_k.$$

Fix any instance J with $J \models \Sigma$ and a homomorphism $h_0 : I_0 \to J$.

Invariant. For every step $I_k \to I_{k+1}$ we can extend $h_k : I_k \to J$ to some $h_{k+1} : I_{k+1} \to J$.

Proof idea: maintain a homomorphism along the chase

Consider a chase sequence (finite or infinite):

$$I_0 \to I_1 \to I_2 \to \cdots \qquad \text{and} \qquad I_\infty = \bigcup_{k \geq 0} I_k.$$

Fix any instance J with $J \models \Sigma$ and a homomorphism $h_0 : I_0 \to J$.

Invariant. For every step $I_k \to I_{k+1}$ we can extend $h_k : I_k \to J$ to some $h_{k+1} : I_{k+1} \to J$.

Then define $h = \bigcup_k h_k$ (well-defined on the union), yielding $h : I_\infty \to J$.

TGD step: extending the homomorphism

Suppose the chase applies a TGD

$$\forall \bar{x}\ (\varphi(\bar{x}) \Rightarrow \exists \bar{y}\ \psi(\bar{x}, \bar{y}))$$

with a match α such that $I_k \models \varphi(\alpha(\bar{x}))$ but not yet $\exists \bar{y}\ \psi(\alpha(\bar{x}), \bar{y})$.

TGD step: extending the homomorphism

Suppose the chase applies a TGD

$$\forall \bar{x}\ (\varphi(\bar{x}) \Rightarrow \exists \bar{y}\ \psi(\bar{x}, \bar{y}))$$

with a match α such that $I_k \models \varphi(\alpha(\bar{x}))$ but not yet $\exists \bar{y}\ \psi(\alpha(\bar{x}), \bar{y})$.

The chase adds fresh nulls $\bar{n}$ and atoms $\psi(\alpha(\bar{x}), \bar{n})$ to get I_{k+1}.

TGD step: extending the homomorphism

Suppose the chase applies a TGD

$$\forall \bar{x}\ (\varphi(\bar{x}) \Rightarrow \exists \bar{y}\ \psi(\bar{x}, \bar{y}))$$

with a match α such that $I_k \models \varphi(\alpha(\bar{x}))$ but not yet $\exists \bar{y}\ \psi(\alpha(\bar{x}), \bar{y})$.

The chase adds fresh nulls $\bar{n}$ and atoms $\psi(\alpha(\bar{x}), \bar{n})$ to get I_{k+1}.

Since h_k is a homomorphism, $J \models \varphi(h_k(\alpha(\bar{x})))$. Because $J \models \Sigma$, there exist witnesses $\bar{b}$ in J with $J \models \psi(h_k(\alpha(\bar{x})), \bar{b})$.

TGD step: extending the homomorphism

Suppose the chase applies a TGD

$$\forall \bar{x}\ (\varphi(\bar{x}) \Rightarrow \exists \bar{y}\ \psi(\bar{x}, \bar{y}))$$

with a match α such that $I_k \models \varphi(\alpha(\bar{x}))$ but not yet $\exists \bar{y}\ \psi(\alpha(\bar{x}), \bar{y})$.

The chase adds fresh nulls $\bar{n}$ and atoms $\psi(\alpha(\bar{x}), \bar{n})$ to get I_{k+1}.

Since h_k is a homomorphism, $J \models \varphi(h_k(\alpha(\bar{x})))$. Because $J \models \Sigma$, there exist witnesses $\bar{b}$ in J with $J \models \psi(h_k(\alpha(\bar{x})), \bar{b})$.

Define h_{k+1} by extending h_k and mapping each new null n_i to b_i. Then h_{k+1} is a homomorphism $I_{k+1} \to J$.

EGD step: compatibility with identifications

Suppose the chase applies an EGD

$$\forall \bar{x} \ (\varphi(\bar{x}) \Rightarrow u = v)$$

with a match α such that $I_k \models \varphi(\alpha(\bar{x}))$, forcing the identification $\alpha(u) \equiv \alpha(v)$ in I_{k+1}.

EGD step: compatibility with identifications

Suppose the chase applies an EGD

$$\forall \bar{x} \ (\varphi(\bar{x}) \Rightarrow u = v)$$

with a match α such that $I_k \models \varphi(\alpha(\bar{x}))$, forcing the identification $\alpha(u) \equiv \alpha(v)$ in I_{k+1}.

Because h_k is a homomorphism, $J \models \varphi(h_k(\alpha(\bar{x})))$. Since $J \models \Sigma$, we must have

$$h_k(\alpha(u)) = h_k(\alpha(v)).$$

EGD step: compatibility with identifications

Suppose the chase applies an EGD

$$\forall \bar{x} \ (\varphi(\bar{x}) \Rightarrow u = v)$$

with a match α such that $I_k \models \varphi(\alpha(\bar{x}))$, forcing the identification $\alpha(u) \equiv \alpha(v)$ in I_{k+1}.

Because h_k is a homomorphism, $J \models \varphi(h_k(\alpha(\bar{x})))$. Since $J \models \Sigma$, we must have

$$h_k(\alpha(u)) = h_k(\alpha(v)).$$

Thus h_k is consistent with the quotient that identifies $\alpha(u)$ and $\alpha(v)$. So it factors through the identification and yields a homomorphism $h_{k+1} : I_{k+1} \to J$.

EGD step: compatibility with identifications

Suppose the chase applies an EGD

$$\forall \bar{x} \ (\varphi(\bar{x}) \Rightarrow u = v)$$

with a match α such that $I_k \models \varphi(\alpha(\bar{x}))$, forcing the identification $\alpha(u) \equiv \alpha(v)$ in I_{k+1}.

Because h_k is a homomorphism, $J \models \varphi(h_k(\alpha(\bar{x})))$. Since $J \models \Sigma$, we must have

$$h_k(\alpha(u)) = h_k(\alpha(v)).$$

Thus h_k is consistent with the quotient that identifies $\alpha(u)$ and $\alpha(v)$. So it factors through the identification and yields a homomorphism $h_{k+1} : I_{k+1} \to J$.

(If your chase detects an EGD contradiction on distinct constants, then there is no solution J; universality is vacuous.)

Conclusion: universal model + CQ answering

By the two extension lemmas (TGD and EGD), the invariant holds for every chase step. Taking the union at the limit gives a homomorphism

$$h : \mathrm{Chase}_\Sigma(I_0) \to J$$

for every $J \models \Sigma$ with $I_0 \to J$.

Conclusion: universal model + CQ answering

By the two extension lemmas (TGD and EGD), the invariant holds for every chase step. Taking the union at the limit gives a homomorphism

$$h : \mathrm{Chase}_\Sigma(I_0) \to J$$

for every $J \models \Sigma$ with $I_0 \to J$.

Therefore $\mathrm{Chase}_\Sigma(I_0)$ is universal.

Conclusion: universal model + CQ answering

By the two extension lemmas (TGD and EGD), the invariant holds for every chase step. Taking the union at the limit gives a homomorphism

$$h : \mathrm{Chase}_\Sigma(I_0) \to J$$

for every $J \models \Sigma$ with $I_0 \to J$.

Therefore $\mathrm{Chase}_\Sigma(I_0)$ is universal.

CQ corollary.

- If $\mathrm{Chase}_\Sigma(I_0) \models q$, then for every solution J, the composition of homomorphisms gives $J \models q$.

Conclusion: universal model + CQ answering

By the two extension lemmas (TGD and EGD), the invariant holds for every chase step. Taking the union at the limit gives a homomorphism

$$h : \mathrm{Chase}_\Sigma(I_0) \to J$$

for every $J \models \Sigma$ with $I_0 \to J$.

Therefore $\mathrm{Chase}_\Sigma(I_0)$ is universal.

CQ corollary.

- If $\mathrm{Chase}_\Sigma(I_0) \models q$, then for every solution J, the composition of homomorphisms gives $J \models q$.
- Conversely, if q is false on the chase, then the chase itself provides a counterexample.

Other applications

- Answering queries over all possible completions of a database constrained by TGDs and EGDs [Calì et al., 2003, 2012]; widely used in Semantic Web reasoning.
- Data integration [Lenzerini, 2002]: TGDs and EGDs express the relations between two schemas, and we query a dataset based on the dependencies and a dataset from the other schema.
- Data exchange [Fagin et al., 2005, Onet, 2013]: TGDs and EGDs express the relations between two schemas, and we use the chase to generate the translation of a dataset into the other schema.
- Recursive queries: TGDs can be used to define queries with recursion (if no $\exists$, this is exactly Datalog, see further lecture). We can reason about the query + constraints using the chase.

Outline

FD and MVD

Chase of FD and MVD

Generalization

General chase

References

References

- The historical paper on the chase [Maier et al., 1979]
- Chapters 8 to 10 of [Abiteboul et al., 1995]
- A modern view of the general chase: [Fagin et al., 2005]

Bibliography I

Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. ISBN 0-201-53771-0. URL `http://www-cse.ucsd.edu/users/vianu/book.html`.

Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011. doi: 10.1016/j.artint.2011.03.002. URL `http://dx.doi.org/10.1016/j.artint.2011.03.002`.

Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In Frank Neven, Catriel Beeri, and Tova Milo, editors, *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 260–271. ACM,

Bibliography II

2003. ISBN 1-58113-670-6. doi: 10.1145/773153.773179. URL http://doi.acm.org/10.1145/773153.773179.

Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, and Andreas Pieris. Datalog+/-: A family of languages for ontology querying. In Oege de Moor, Georg Gottlob, Tim Furche, and Andrew Jon Sellers, editors, *Datalog Reloaded - First International Workshop, Datalog 2010, Oxford, UK, March 16-19, 2010. Revised Selected Papers*, volume 6702 of *Lecture Notes in Computer Science*, pages 351–368. Springer, 2010. ISBN 978-3-642-24205-2. doi: 10.1007/978-3-642-24206-9_20. URL http://dx.doi.org/10.1007/978-3-642-24206-9_20.

Bibliography III

Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012. doi: 10.1016/j.artint.2012.08.002. URL `http://dx.doi.org/10.1016/j.artint.2012.08.002`.

Ashok K. Chandra, Harry R. Lewis, and Johann A. Makowsky. Embedded implicational dependencies and their inference problem. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA*, pages 342–354. ACM, 1981. doi: 10.1145/800076.802488. URL `http://doi.acm.org/10.1145/800076.802488`.

Bibliography IV

Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005. doi: 10.1016/j.tcs.2004.10.033. URL `http://dx.doi.org/10.1016/j.tcs.2004.10.033`.

Maurizio Lenzerini. Data integration: A theoretical perspective. In Lucian Popa, Serge Abiteboul, and Phokion G. Kolaitis, editors, *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 233–246. ACM, 2002. ISBN 1-58113-507-6. doi: 10.1145/543613.543644. URL `http://doi.acm.org/10.1145/543613.543644`.

Bibliography V

David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979. doi: 10.1145/320107.320115. URL `http://doi.acm.org/10.1145/320107.320115`.

Adrian Onet. The chase procedure and its applications in data exchange. In Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt, editors, *Data Exchange, Integration, and Streams*, volume 5 of *Dagstuhl Follow-Ups*, pages 1–37. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. ISBN 978-3-939897-61-3. doi: 10.4230/DFU.Vol5.10452.1. URL `http://dx.doi.org/10.4230/DFU.Vol5.10452.1`.